APHIDS++: Evolution of A Programmable Hybrid Intrusion Detection System

by

Mohammed Shahidul Alam

B.Sc., The University of Texas at Austin, 2002

A THESIS SUBMITTED IN PARTIAL FULFILMENT OF
THE REQUIREMENTS FOR THE DEGREE OF

Master of Science

in

The Faculty of Graduate Studies

(Computer Science)

The University Of British Columbia

October, 2005

© Mohammed Shahidul Alam 2005

Abstract

With the rapid growth of the Internet and the ever-increasing security problems associated with its popularity, the need for protection against unwanted intruders has become essential. Antivirus software, intrusion detection systems, spyware and malware detectors are some of the protection mechanisms available to users today. The diversity of these manifold systems suggests the need for a unifying managerial system, such as APHIDS (A Programmable Hybrid Intrusion Detection System), a mobile agent based IDS, which can correlate and coalesce preexisting security components. In this thesis we provide a description of improvements made to the initial APHIDS design, comprising the addition of an optional intelligent agent meant to improve the response of APHIDS in detecting VoIP (Voice over IP) and generic intrusions; and an XML implementation of our Agent Deployment and Correlation Script (ADCS), which is used to initialize the agent environment, allowing for flexible user modifications to control the deployment and invocation of mobile agents.

Table of Contents

Abstract	ii
Table of Contents	iii
List of Tables	vi
List of Figures	vii
Acknowledgements	x
1 Introduction	1
1.1 Motivation	1
1.2 Thesis Contribution	2
1.3 Structure of Thesis	3
2 Background work	4
2.1 Agents	4
2.1.1 Agent Classification	4
2.1.2 Agent Design Paradigm	5
2.1.3 Advantages and Disadvantages of Agents	6
2.2 Intrusion Detection Systems	8
2.2.1 IDS Classification	8
2.2.1.1 Host-based intrusion detection systems	9
2.2.1.2 Network-based intrusion detection systems	10
2.2.1.3 Misuse-based intrusion detection systems	11
2.2.1.4 Anomaly-based intrusion detection systems	11
2.2.2 Techniques and Tools used by intrusion detection systems	12
2.2.2.1 IDS Correlation	12
2.2.2.2 Testing Intrusion Detection Systems	13
2.2.2.3 IDS Visualization	13
2.3 Agent-based Intrusion Detection Systems	13

2.4	Initial APHIDS design	15
2.4.1	APHIDS objectives	15
2.4.2	Logical Phases in APHIDS	16
2.4.3	APHIDS components	16
2.4.3.1	Correlation Routine	18
2.5	Attack Languages	18
3	APHIDS++: Policy Agent	21
3.1	Need for Policy Agent	21
3.2	APHIDS++ Agent Interaction	21
3.3	Policy Agent Design Decisions	23
3.4	Implementation Details	23
3.4.1	Grasshopper Environment	23
3.4.2	Deployment	25
3.4.3	Changes to Trigger and Correlation Agent	25
3.4.4	Communication with Policy Agent	25
3.5	Evaluation of Policy Agent	26
3.5.1	DoS attack on APHIDS++	26
3.5.1.1	Test Environment Setup	26
3.5.1.2	Test Methodology	26
3.5.1.3	Test Results	29
3.5.1.4	Policy Agent introducing more latency	31
3.5.2	Detecting VoIP attack on H.248/Megaco system	31
3.5.2.1	User Profiling	31
3.5.2.2	Test Environment Setup for detecting VoIP Call Hijack Attack	33
3.5.2.3	Test Methodology for detecting VoIP Call Hijack Attack	34
4	APHIDS++: ADCS	37
4.1	ADCS categorization	37
4.2	Design decision to use XML	38
4.3	XML related work in mobile agents	39
4.4	ADCS Implementation	39
4.4.1	XML Parsing Basics	39
4.4.2	Xerces XML for java library	40
4.4.3	Description of XML Syntax and rules	40
4.4.3.1	Condition Evaluation	46
4.5	Evaluation of XML-based ADCS	47
4.5.1	Multiple correlation routines	47

4.5.2	Proper invocation of Policy Agent	48
4.5.3	Proper invocation of Trigger Agent	48
4.5.4	Proper Evaluation of Rules	49
5	Conclusion and Future Work	51
5.1	Improvements to APHIDS++	52
5.2	Improvements to Policy Agent	53
5.3	Improvements to Agent Deployment and Correlation Scripts .	53
	Bibliography	55
A	APHIDS++ API	61
A.1	CorrelationObject API	61
A.2	Policy Agent API	62
B	Call Hijack Simulation	63

List of Tables

- 3.1 This table shows the configuration of the hardware used to
compare the DDoS test performed on APHIDS++ and APHIDS 27
- 3.2 This table shows the latency period obtained from tests based
on the number of portscans done on APHIDS and APHIDS++ 30

List of Figures

2.1	This figure shows the structure of a network that deploys Host-Based IDS systems only	9
2.2	This figure shows the structure of a network deploying Network-based and Host-based IDS systems	10
2.3	This figure shows the interaction between all the agents in the initial APHIDS design	17
2.4	This figure shows a birds-eye view of the interaction between the various agent types in a correlation routine	19
3.1	Interaction between agents with the presence of Policy Agent in APHIDS++	22
3.2	This figure shows the presence of agents deployed using Grasshopper agent environment	24
3.3	This figure presents the setup used to compare APHIDS and APHIDS++ latency measurements	28
3.4	This figure shows a log scale comparison of the latency measurements obtained from portscan tests in APHIDS++ and APHIDS	30
3.5	Use of APHIDS++ in a VoIP call hijack attack scenario . . .	34
4.1	XML code showing the description of a correlation routine . .	41
4.2	This figure shows the structure of the HashMap generated for a correlation routine after reading the XML-based ADCS . .	42
4.3	This figure shows the structure of the HashMap generated for the ruleSet variable of a correlation routine	43
4.4	This shows the usage of the allRoutines tag to encapsulate all the correlation routines in ADCS	43
4.5	This shows the usage of the correlationRoutine tag with its list of attributes	44
4.6	This shows the usage of the ruleSet tag used to encapsulate the set of rules of a correlation routine	45

4.7	This shows the usage of the rule tag with its set of attributes and nested elements	46
4.8	This shows the use of the 'list' command for checking that multiple correlation routines were successfully read from the ADCS	47
4.9	A message is displayed to the user by the Trigger Agent after it gets data from the Policy Agent showing proper Policy Agent was invoked	48
4.10	Test to show the Trigger Agent was invoked properly from within the script: The Trigger Agent is contacting the Policy Agent to check if the sourceIP of the orphaned packet is in the allowed list of IPs that can use this MG	49
4.11	Test to show the proper invocation and evaluation of rules . .	49
4.12	Test to show the proper evaluation of rules as seen in the APHIDS++ UI	50
B.1	A user has to first enter the web address where the APHIDS++ class files can be found	63
B.2	A user can enter the <list> command to see if the XML file was read correctly	64
B.3	A user has to enter the enable <correlationRoutine> command to start the correlation task	65
B.4	This shows that correlationObjects abstract class policyEngineOn() has been executed. This is an important class since if the return value is true, then the Policy Agent is enabled, else it is not	65
B.5	The user now has to enter the location where the Policy Agent will reside	66
B.6	The user now has to enter the location of the Trigger Agent which sits in the machine where MG (media gateway) and MG.log exists	66
B.7	Enter the location of where the MG.log file exists	66
B.8	The user now has to enter the location of the Task Agent which will move to the machine where MGC (media gateway controller) and MGC.log exists	66
B.9	Enter the location where the MGC.log file exists	67
B.10	A request is made to the user to allow B.9 and B.10 to take place before he/she can click on OK. This is because step B.8 happens in parallel to B.10. B8 and B9 have to execute first for the system to function properly	67

B.11	The user has to enter the simulated orphaned packet information. For the system to work properly, the SrcIP has to exist in the allowed list of the Policy Agent. The SrcIP cannot exist in MG.log (which keeps track of active connections), since that would mean that it is not an orphaned packet. The destinationIP has to exist in MGC.log with the REINVITE message flag	67
B.12	The Trigger Agent is contacting the Policy Agent to check if the sourceIP of the orphaned packet is in the allowed list of IPs that can use this MG	68
B.13	A message is displayed to the user mentioning that this is a valid source IP for an orphaned packet. The trigger agent is now going to conduct a correlation operation to find the user who sent the REINVITE message	68
B.14	The correlation Agent will make a call to the MachineLearning component of the Policy Agent. We do this step just to check if we can contact the MachineLearning component . . .	68
B.15	This message reconfirms that the correlation Agent successfully contacted policy Agents Machine Learning component .	69
B.16	At this point the correlation Routine has already contacted the Task Agent to move to the MGC to get relevant data from MGC.log regarding the intrusion. Once it gets the data from the Task agent, it contacts the policy Agent to return the profile of the intruder. The correlation routine then forwards the profile to the action agent to display the information on the console	69
B.17	The final message shows the results of the correlation process in the UI	70

Acknowledgements

I would like to thank Dr. Son Vuong for his helpful supervision over the last couple of years and the second reader of this thesis, Dr. Charles Krasic for being available to review this thesis in a short time frame.

I would also like to thank my parents because of whom I am where I am today. Special thanks goes to all those with whom I had the opportunity to interact with over the last couple of years including Christian, Kapil and Ken.

This research is supported in part by the Directorate of Telecom Engineering and Certification of the Department of Industry Canada.

Chapter 1

Introduction

Intrusion detection systems (IDS) are rapidly being deployed in both industry and academia to provide the best possible protection of personal and confidential information. Because different intrusion detection systems are good at detecting diverse kinds of attacks, it makes sense to combine the strengths of various existing IDSs to form a general, overarching security solution. This was the main philosophy underlying the original APHIDS system [16]. This thesis reports our contributions in extending the original APHIDS architecture, now called APHIDS++¹. Notable extensions include the incorporation of an optional intelligent agent, called the Policy Agent, that allows the system to learn from previous intrusion attempts; and the incorporation of an XML based correlation scripting language to initialize and control the entire APHIDS++ system.

1.1 Motivation

With the rapid growth of the Internet and the ever-increasing security problems associated with its popularity, the need for protection against unwanted intruders has become imperative. An unwanted intruder could be anyone. It could be an outsider trying to hack a database system to access available credit card information of valuable clients of a company; or it could be a close friend trying to access personal information stored in a friend's personal diary without his or her authorization. Hence the term *unwanted intruder* depends on the *context* of the information that is being accessed.

Intrusion detection, as the name suggests, is the process utilized to detect the presence or occurrence of an intrusion. Some of the questions that an intrusion detection system tries to deduce include, what harm has the intruder already done? What information can he access now? Was the intruder successful in his intrusion attempt? And if he was, is the system prone to a similar intrusion in the future? These are all the questions that an intrusion detection system tries to answer in one way or another. How

¹components of this thesis work has been accepted for publication [1]

effective they are in their detection is another story.

There are a lot of variables that go into evaluating the effectiveness of an intrusion detection system. One of the elements that an intrusion detection system is judged on includes its effectiveness in detecting most intrusions correctly i.e. an IDS is not expected to detect each and every intrusion as this is logically infeasible. Other factors that go into evaluating IDS efficiency include keeping the number of false positives (the number of cases that are reported as intrusions but are not really intrusions) at a minimum and its ability to respond quickly in detecting attacks.

The reason why we worked on improving our base system (APHIDS) is because though it could detect intrusions effectively depending on the type of intrusion, it was slow in completing the intrusion response task. To elaborate, given a particular detection task, the system did not learn from a previous intrusion attempt of the same type and origin. This would cause the system to solve the same problem it had solved previously. This made the system prone to denial of service (DoS) attacks by intruders who could send the same attack type in quick succession in a very short frame of time. In order to solve this problem we introduced a feedback loop in the form of our intelligent Policy Agent. This Policy Agent could be used using multiple detection techniques to allow the system to keep learning and increasing its ability to perform complicated tasks with a low overhead and latency.

1.2 Thesis Contribution

The following are the main contributions of this thesis as they appear in the subsequent sections:

1. The first and foremost contribution is the design and implementation of an intelligent Policy Agent that allows the system to learn from previous intrusion attempts.
2. After building the prototype Policy Agent, we then apply it to an intrusion detection scenario that was conducted in the base APHIDS system, which has been extended in this thesis.
3. We then provide an empirical comparison of the results obtained in tests conducted using the introductory system developed by Deeter [15] and the evolved system (APHIDS++).
4. We emulate a system that demonstrates the effectiveness of applying the Policy Agent to detect Voice over Internet Protocol (VoIP) intru-

sions. This was done by testing for deviation from user profile using simple misuse detection techniques, thus managing to reduce the number of false alarms that would have otherwise been introduced into the system.

5. Finally we introduce an initial prototype of our XML-based Agent Deployment and Correlation Script (ADCS) that allows for system initialization and control of rules for proper agent deployment and response.

1.3 Structure of Thesis

This thesis is broken down into the following subsequent chapters. Chapter 2 provides a brief introduction to mobile agents, taxonomy of intrusion detection, and the categorization of different types of attack languages used in intrusion detection systems. Chapter 2 also provides a background on the important basic features of APHIDS to allow one to follow the evolution that the system has undergone. Chapter 3 discusses the architectural changes that were made to the base APHIDS system in the form of the intelligent Policy Agent. We provide design details and the empirical test evaluations conducted using a controlled test environment. In Chapter 4 we discuss the initial prototype of the SAX (Simple API for XML) based XML, and its usage in our system. Chapter 5 provides the conclusion and a list of improvements that can be made to the current APHIDS++ system before we can attempt to challenge commercial intrusion detection systems.

Chapter 2

Background work

In order to better comprehend the context of this thesis work, the reader needs to understand a few of the related work that is available in literature pertaining to this thesis. This section discusses four main topics of interest. This includes a brief overview of mobile agents, taxonomy of intrusion detection systems, the languages used to control and deploy intrusion detection systems, and a brief overview of the base APHIDS system that has been extended and improved upon in this thesis.

2.1 Agents

The need for a clear understanding of agents is necessitated due to the fact that the intrusion detection system that we have developed and extended, is layered on top of an agent platform, called *Grasshopper*[9], based on the first mobile agent standard MASIF² (Mobile Agent System Interoperability Facility) of OMG (Object Management Group). The term agent or software agent is usually deciphered well in the artificial intelligence community, where it stands for a program that can behave autonomously to perform a multitude of dynamic tasks based on the logistics that have been programmed into it by a user.

2.1.1 Agent Classification

Based on the mobility of agents, they can be classified into three main types:

- *Static Agents*: The first is the concept of static agents. Static agents are a fragment of code that do not move to different locations, and stay at a constant position throughout its life cycle i.e. they remain at the same logical and physical location from the point of creation to the point when they are destroyed, or the program terminates.

²MASIF is a interoperability standard that allows agents from different mobile agent platforms to interact with each other

- *Semi-Mobile Agents*: Semi-mobile agents, as the name suggests, have some mobility. They are in fact an inherent type of mobile agents, which are created at one logical or physical location, but are moved to another location for its functional life cycle.
- *Mobile Agents*: Mobile agents are a fragment of code, which can move around, hopping from machine to machine during its life cycle depending on the runtime task allocated to it. Mobile agents are based on a terminology, well known in literature as *mobile code* [23]. The term mobile code can be defined as the capability to change the binding between the pieces of code, and the location where they are executed.

The scope of the advantages or disadvantages of using any of the above mentioned agent types can vary based on the functionality of the agent based system that is being deployed. If latency is a big issue in the system, one should opt for static and/or semi-mobile agents. This is because the greater the mobility of an agent, the higher the latency introduced into the system caused by the time required to create it at a new location and to transfer the runtime state of the agent.

If the host where the agent runs is very fragile or more prone to destruction or tampering, it would be best to use a mobile agent rather than a static agent, as it is easier for mobile agents to find a new location to run at than static agents.

2.1.2 Agent Design Paradigm

As mentioned by Fuggetta et al. in [23], it is important to understand the design paradigm to be used while using mobile agents. Some of the cost benefits that have to be measured include latency, memory access, partial failures and concurrency. The following are some of the design paradigms that can be used with mobile agents:

- *Client-Server model* In a client-server model, a client at location *A*, asks a server at location *B* to perform a certain task. *A* does not have the means or methods of processing a given task, but can only make request for work to be done. An example of this is a Java RMI (Remote Method Invocation) call.
- *Mobile agent (MA) model* In a Mobile agent model, *user-x* may have some means and methods to process data at location *A*. There are also data or methods located at another location *B* that *user-x* requires for

completing the process. In that case, *user-x* can migrate from location *A* to location *B*, carrying with it some of the processed data from *A* and the know-how to process the data available at *B*.

In APHIDS, the two design paradigms used are the client-server and the mobile agent model. Hence the readers are referred to [23] for a further explanation of the remaining two models namely, *Remote Evaluation (REV)* and *Code on Demand (COD)*.

2.1.3 Advantages and Disadvantages of Agents

The use of mobile agents offers wide advantages specially in distributed systems that cannot be overlooked. Some of the advantages offered by agents have been clearly listed in [10]. The major categories of these are summarized as follows:

- *Reduction in Network Traffic:* As we know, in case of mobile agents, the agents themselves move to data. i.e. we move the agent *code* to the data rather than moving the data to the *agent code*. This allows for a dramatic reduction in the amount of bandwidth consumed in the log correlation process (explained in later sections) as data is almost always larger than the few Kb size of agents in general.
- *Asynchronous autonomous interaction:* An advantage of mobile agents is its ability to asynchronously process information. This is vital in a network where network connections are volatile, such as wireless networks. In such cases, the agent could migrate to a mobile device to gather data. Even if the connection breaks, the agent could continue processing data on the mobile device and report back whenever the connection is reestablished. This adds to the agent's capability to work in a fault tolerant mode.
- *Software Upgrades:* Usually in order to update software on multiple hosts, an administrator has to first stop the server functionality, then uninstall the old version of the software, and then reinstall the new version. The entire software system has to be stopped for upgrades. The advantage of mobile agents or agents in general in this situation is that if each component of the upgraded software is managed by an agent, then it is as easy as disabling the old agent and deploying a new agent which has the required functionality. In this way one could avoid bringing down the entire system and instead stop just a single agent-based component.

- *Functionality in heterogeneous environments:* Most agents today can work in heterogeneous environments. This is due to the fact that these agents are usually written in a language which is portable to multiple platforms, such as java or perl. Since agents sit on top of an agent framework, they can easily function regardless of if the host runs a version of Linux or Windows operating system. The significant reduction in costs of placing agent frameworks in hosts over the past few years have added to the benefits of running agents.

Just like there are advantages to using agents, there are also drawbacks to using agents. The applicability of advantages or disadvantages to using agents is based immensely on the specific user needs or goals that have been put forward. The shortcomings of using a mobile agent-based system has been clearly summarized by Vigna in [46]. Some of the major drawbacks mentioned by authors include:

- *Agent Security* The one and only reason that has hindered the wide usage of mobile agents in the real world has been its security constraints. One of the key problems associated with mobile agent security is the *malicious host* problem i.e. how much trust can be placed on a host where the agent travels to, given that the agent may have valuable highly secured data. This data could be as vital as a persons credit card information in an unencrypted format, or the password to ones bank account. Many have claimed that if the agent is placed in a closed environment [15] then this problem does not exist. But the fact is that this problem still persists in situations when an intruder has overtaken a system in a closed environment without the knowledge of the administrator.

Other security concerns that have been mentioned in literature include the concept of *malicious agents* [47] where given the availability of an agent platform in a host, how much trust can be placed on the agent that travels to the host to gather information? This problem has been solved in the agent-security field by allowing a host to run only certain digitally signed agents.

Last but not the least, agents can be *tampered* with which means, a legitimate agent could be *brainwashed* [46] while traveling from host to host. Vigna has provided a means for auditing an agents trail to detect attacks that modify agents legitimate access permissions and authorization mechanisms for the aglets mobile agent platform [47].

- *Lack of Shared Language* Even though many tasks have been overtaken by FIPA (The Foundation for Intelligent Physical Agents) to create a standard ACL (Agent communication language) ³, most agent platforms do not adhere to this language. Hence it is hard for agents to communicate with each other when they are based on different platforms.
- *Required Agent Platform* Any piece of agent code available today needs to run on an agent platform that contributes to the control and deployment of agents. For example, our APHIDS system has to use the Grasshopper agent platform to execute its tasks. Similarly, to run java applets, the system has to have a java runtime environment available. The dependence of mobile agents on an agent platform is an extra requirement that has to be made, without which they cannot function. The problem is further compounded by the fact that not all agent platforms follow a given set of rules and procedures thus hindering interoperability issues even with the existence of standards such as MASIF (explained previously).
- *Denial of Service* Any piece of code that is written by a programmer can have flaws. For example a user could perform a logical error in his code by making the *fork()* system call in a while loop. The presence of such snippets of code in a mobile agent code that travels to a location and executes it, could allow them to launch denial of service attack against the host where they reside by hogging all the available system resources [25].

2.2 Intrusion Detection Systems

As mentioned previously intrusion detection is the process utilized to detect the presence or occurrence of an intrusion. Intrusion Detection System (IDS) is an essential tool that compliments any security mechanism such as a firewall or an antivirus. Today most of the surveillance and security monitoring of network infrastructures is done using IDSs.

2.2.1 IDS Classification

Intrusion detection systems, like any set of systems, can be classified. The two main ways that they are classified include:

³a draft is available at <http://www.fipa.org/repository/aclspecs.html>

Figure 2.1: This figure shows the structure of a network that deploys Host-Based IDS systems only

1. Host Based Systems Vs. Network Based Systems
2. Misuse Based Systems Vs. Anomaly Based Systems

We now attempt to describe the functionality of each of these types of systems in turn.

2.2.1.1 Host-based intrusion detection systems

Host-based intrusion detection systems directly monitor the hosts on which they run. These systems can monitor at different levels in a host. It could be at the network level, at the operating system level or at the application level. This is in sharp contrast to Network-based IDSs which may not be able to detect a malicious attack spread across multiple packets or attacks placed in an encrypted format in the payload of a packet. Some of the

commercial and open source host based intrusion detection systems include AIDE (Advanced Intrusion Detection Environment) and Tripwire. The only drawback of an HIDS is that it has to be individually installed in each host, making the task cumbersome on administrators. A typical HIDS system has a network structure as shown in Figure 2.1. The BlueBox system [12] is a host based intrusion detection system that works at the kernel level using system call introspection. The *Backtracker* system [26] is a recent host-based IDS that backtracks an intrusion from the point of detection to the point of source within a host by creating a chain of processes used by the attacker. This system helps a system administrator to detect *zero-day* attacks and apply the necessary patches on the jeopardized hosts.

2.2.1.2 Network-based intrusion detection systems

Figure 2.2: This figure shows the structure of a network deploying Network-based and Host-based IDS systems

Network-based IDSs, as the name suggest, usually monitor all incoming

packets into the network by residing on the path of incoming traffic, as this is the main source of intrusions. For example, in Figure 2.2 the NIDS1 component tracks all incoming packets from the internet and blocks the source IP of the intruder if it finds a malicious packet. The simplest form of NIDS inspection matches string patterns against the payload of packets captured on the network link. Due to the wide use of wireless devices that are physically brought into the inner perimeters of a network, and connected to a secured network, placement of devices such as NIDS1 would not help as it does not detect packets flowing from a malicious internal device to other internal devices. To handle this case, equipments such as NIDS2 shown in Figure 2.2 are installed into the network to track internal malicious packets in promiscuous mode. Such systems include the popular ARGUS system and the libpcap-based packet sniffer and logger SNORT system [39] which is a lightweight version of the commercially available NFR (Network Flight Recorder). Many projects such as the one by Antonatos et al. [4] have tried to evaluate the effectiveness of NIDS by generating realistic workloads by simulating DNS, SMTP and HTTP packets. Vigna et al. have proposed a network based misuse system using the NetSTAT system [49] that detects intrusions using the STAT language. A byte-level NIDS [42] based on anomaly detection, which looks at context of the attack string rather than simple byte sequencing of attack strings has been proposed for reducing the number of false positives.

2.2.1.3 Misuse-based intrusion detection systems

Misuse based systems are also known as signature based systems. SNORT [39] is one of the most popular misuse-based systems. These systems usually act in a similar manner to anti viruses and are designed to look for known, suspicious pattern in incoming data. The main advantage of misuse bases systems is that they suffer from virtually no false positives (i.e. they do not classify non intrusive behavior as an intrusion). However, since these systems can track just known intrusion signatures, these systems suffer from the serious disadvantage of not being able to detect variations of known attacks or entirely new intrusions (zero-day attacks).

2.2.1.4 Anomaly-based intrusion detection systems

A new generation of intrusion detection systems now focus on *anomaly* detection. Anomaly detection systems model normal or expected behavior of a system, and then try to detect deviations from normal which in turn would

indicate a probable intrusion attempt.

Host-based anomaly detections can be used to monitor user or program behavior in a system. One system that has worked on building user profiles based on anomaly detection techniques, such as program and keystroke speed monitoring includes the NIDES system [3].

Network-based anomaly detection systems such as the adhoc system by Huang et al. [2] usually model the flow of packets by using the source and destination IPs, ports and the number of connections in a given period of time. Recent network-based anomaly detection techniques used include that by Wang et al. [50] where a feature vector is constructed and *Mahalanobis* distance algorithm is used to detect anomalous behavior of network packet content on the 1999 DARPA IDS dataset [33]. An in-depth benchmarking technique for anomaly detection systems have been proposed by Maxion et al. in [32] so that one can compare the effective of various anomaly detection systems. Sommer et al. in [42] have built an anomaly based NIDS by using Snort [39] logs and converting them into Bro language [37] expressions.

2.2.2 Techniques and Tools used by intrusion detection systems

One of the most widely deployed intrusion detection system is SNORT [39], a *misuse* detection system that has been used in the APHIDS system described later. An IDS functions by first generating a log to keep record of an attack once it detects an intrusion. After that, an IRS (Intrusion Response System) takes over. Toth et al. have presented an automated response system [44] that allows administrators to automate the response task. This is helpful in cases when an IDS generates more alerts than a human can handle.

2.2.2.1 IDS Correlation

Most intrusion detection systems today use the process of *log correlation*, which is a process that takes the alerts generated by multiple intrusion detection systems and produce a brief report on the network being protected [45].

The advantage of this method is that if there are multiple intrusion detector sensors deployed in the network, on the occurrence of an intrusion attack, each of these sensors would generate a report on the intrusion type. Allowing log correlation of the information generated by all these sensors would provide a system administrator with a compact but detailed report on the attack allowing him or her to pinpoint the vulnerability easily.

In the conventional log correlation model, distributed sensors, after gathering the data, send all the alerts to a centralized location for correlation purposes. But the major disadvantage of this model is that if the amount of logs generated is large, it would clog the network system in a low-bandwidth network. Also a centralized approach would overload a node that receives too many correlation tasks at a given time, causing system overload and hence delay in producing the analyzed results.

The most detailed work done to date is by Valeur et al. [45] that provides a basic framework required by log correlation based intrusion detection systems. In it the authors had tested their multi-step log correlator on the Defcon 9⁴ data set using Snort, WinSTAT and USTAT as the sensors.

2.2.2.2 Testing Intrusion Detection Systems

A number of research works have focused on testing and comparing intrusion detections systems mentioned in the earlier sections. Hugh et al.[33] critiques the 1998 and 1999 DARPA IDS evaluations conducted by Lincoln labs in [30] and question there use of ROC (Relative Operating Characteristic) curve to measure the effectiveness of an IDS. Lipmann et al. [30] compare IDSs based on different operating systems and their effectiveness in detecting different attack types such as Denial of Service [35], User to Root and Surveillance intrusion attacks.

2.2.2.3 IDS Visualization

Due to the huge number of alarms generated in busy networks, a considerable amount of work has gone into visualizing large IDS data sets. Barford et al. [8] tries to characterize network traffic flow anomalies and visualizes flash crowd, network abuse (using portscans) and bandwidth usage anomalies. Erbacher et al. [20] tries to visualize log files generated by the *Hummer* intrusion detection system to visualize user behavior. Teoh et al. [36] tries to visualize BGP(Border Gateway Protocol) data to detect anomalies in network routing patterns.

2.3 Agent-based Intrusion Detection Systems

Now that we have explained both the concepts of mobile agents and intrusion detection, it would be useful to cover earlier research projects that have

⁴Defcon is a underground hacking conference held every year.

focused on merging the advantages of both these areas, an area which the APHIDS system can be directly placed under as APHIDS uses an agent-based intrusion detection model.

The earliest relevant work was started by Purdue University's CERIAS (The Center for Education and Research in Information Assurance and Security) group in 1995 when they put forward a proposal for building an autonomous agent based security model by using genetic programming [13]. This was followed up by their work in implementing the earlier proposal found in [13] in 1998 by Balasubramaniyan et al. [7]. This system was called AAFID (Autonomous Agents for Intrusion Detection) written earlier in Perl, Tcl/Tk and C, and later revised and written in the perl language to make it more portable. Another work done in 1998 by Helmer et al. in [24] used an anomaly detection technique by using the *Ripper* algorithm on *sendmail* system calls. The architecture mimicked a portion of the Java Agents for Meta-Learning (JAM) project [43].

A distributed hierarchical IDS was proposed by Mell et al. in [34] that tries to randomize the location of agents and decentralizing directory services. The system also resurrects agents killed by an intruder as there always exists multiple copies that track the original agent and vice versa. The *Micael* IDS was proposed in 1999 [18]. This system shares features similar to APHIDS with the exception that this was just a proposed system with no proof as to its actual implementation. They do however propose an additional feature of periodically checking if all agents are alive and well in the system.

Another prominent work that detects intrusions using mobile agents is the IDA system [5]. This system tries to backtrack intrusion attempts by looking into MLSI (Mark Left by Suspected Intruders) left at each host. They also emphasize the steps that an attacker takes. The steps taken by intruders are broken down into *search* stage where the intruder looks for vulnerabilities in a systems, *activity* stage where the attacker attempts to break in, *mark* stage where the attacker leaves a mark, such as modifying a password file and finally *masquerade* stage where the attacker tries to cover up modifications he made to administrative files and uses a user account to hide his or her actions.

The *Sparta* system by Kruegel et al. [27][28] is the most extensive work done till date on using mobile agents and intrusion detection. Sparta, which stands for Security Policy Adaptation Reinforced Through Agents, is an architecture that is capable of monitoring a network to detect intrusions and security policy violations by providing a query like functionality to reconstruct patterns of events across multiple hosts. This is a network-based IDS

that *correlates* data from multiple sensors located throughout the network. The authors have created an EQL (Event Query Language) with syntax similar to SQL (Sequence Query Language) used in databases.

Other mobile agent based IDS's include a P2P based IDS [38] that works in a *neighborhood watch* manner where each agent looks after other agents in its vicinity by using a voting procedure to take action against a compromised agent; the MA-IDS system [29] also has a structure similar to APHIDS with the exception that they use encrypted communication between the mobile agents in the system, plus they use a threshold mechanism to detect the probability for each intrusion depending on the quantity of each intrusion type obtained allowing it to learn in a one dimensional method. Some other mobile agent based IDS's include a position paper [6] that claims to work on D'Agents environment; and work by Foukia et al. in [21, 22] which uses a social insect metaphor and immune systems to model an intrusion detection system.

2.4 Initial APHIDS design

Since the main focus of this thesis is in extending the original APHIDS architecture constructed by Deeter in [15], it is imperative that the readers are exposed to the basic APHIDS framework and its functionality.

Perhaps the most salient aspect of APHIDS, and the mechanism by which it can out-perform conventional IDS correlation techniques, is its use of mobile agents. The correlation of IDSs often involves mining large repositories of data stored locally at IDS machines. Mobile agents allow APHIDS to perform such investigations on-site, thereby obviating the need to transmit large quantities of data across the network. A single programmed agent can thus evaluate large data logs at a number of different IDSs simply by migrating from machine to machine as required. This very idea is the key motivation behind the APHIDS architecture. In effect, the APHIDS system, implemented in Java atop the mobile agent platform Grasshopper, is a mobile agent based intrusion detection framework which integrates existing IDS technologies to retrieve more detailed information about an intrusion.

2.4.1 APHIDS objectives

The main objective of the initial APHIDS design was to design a mobile agent based system that handles some of the limitations that centralized detection based systems have. Namely, distributing the workload and avoiding

a single point of failure. With these objectives in mind the following were some of the objectives displayed by APHIDS:

- *Reuse detection logs*: One of the main features of APHIDS is its ability to explicitly reuse log files generated by other IDSs. For example, in the case of APHIDS, log files from SNORT were parsed to detect portscan operations by an attacker. APHIDS also used SSH log files to correlate the attacker source once it detected an intrusion attempt.
- *Distributed Analysis*: In APHIDS the analysis task is distributed. For example, instead of bringing the log files of SNORT and the SSH server to a centralized location for processing, mobile agents traveled to each server to gather and then filter only the required information.
- *Modular design*: APHIDS follows a very modular design that allows for easy addition of components and extensions to the APHIDS framework. This was one reason why it was easier for us to extend the initial APHIDS design. A modular design was followed in APHIDS to allow for easy *integration* with other IDSs, *extensibility* to allow for modified pattern detection techniques without modifying the basic architecture of the system and *programmability* that allows users to reuse parts of the system for new attack detection and correlations.

2.4.2 Logical Phases in APHIDS

The APHIDS architecture consists of three logical phases that is used to detect and perform an intrusion detection cycle. The first phase is the *detection* phase where an intrusion is detected by the aid of a highly-configurable mobile agent based sensor. This is followed by the *correlation* phase during which the system gathers data from multiple sites before intelligently correlating the gathered information. The third and final stage is the *action* phase during which the system takes appropriate actions in case the correlation process detects a probable intrusion. This action could be to notify administrators or changing router rules to block incoming traffic from an intruder IP.

2.4.3 APHIDS components

In this section, we provide a birds eye view of the interconnectivity among the various agent types in APHIDS required for a clear understanding in the next chapter, without going into the nitty-gritty details of the interaction and functionality of each component.

Figure 2.3: This figure shows the interaction between all the agents in the initial APHIDS design

As shown in Figure 2.3, APHIDS is actually a combination of different agent types integrated with each other in a modular design. The functionality of each agent type can be described as follows:

- *User Interface (UI) Agent:* This agent is started once at system initialization. The main objective of this agent is to provide a user interface to control and view results of intrusions. A user uses the interface shown by this agent to start, stop and query various correlation agents in the system.
- *System Agent:* The System Agent accepts commands entered by the user from the UI Agent, and uses the commands to control and manage all the active and inactive correlation routines in the system.
- *Trigger Agent:* A Trigger Agent is the one which monitors and detects intrusions in the system. Once a Trigger Agent detects an intrusion, it contacts the Correlation Agent and passes vital information regarding the intrusion to it.
- *Correlation Agent:* This agent is responsible for receiving Trigger data from the Trigger Agent. It is responsible for gathering data from other

sources by launching Task Agents and then correlates all the gathered data to decide on if an intrusion should be reported to the Action Agent.

- *Task Agent*: The Task Agent is launched by the Correlation Agent. This is a true mobile agent in the sense that it actually travels to one or more locations to gather relevant data, and then sends it back to the Correlation Agent for correlation purposes.
- *Action Agent*: Once a Correlation Agent decides that an intrusion has occurred, it invokes the Action Agent. The Action agents can report all the correlated information to the UI Agent for displaying purposes through the System Agent, or it could travel to a router to modify its access control list.

2.4.3.1 Correlation Routine

A *correlation routine* is the most important logical structure of the APHIDS system. Figure 2.4 shows the various components of a correlation routine. In APHIDS, a correlation routine is described as the entire analysis cycle, and the various agent types used from the point of detection using the *trigger agent* to the point that the final action using the *action agent* is taken. Once a user enters the *enable <correlationRoutine>* command from within the UI agent in APHIDS, the entire set of trigger agent, task agent, correlation agent and action agents are initialized and deployed. There could exist a maximum of one trigger agent and correlation agent, and unlimited number of task and action agents. A user has the ability to mix and match previously developed trigger, task and action agents to develop a new correlation routine. The only component that has to be reprogrammed is the logistics of the correlation agent.

2.5 Attack Languages

Intrusion detection systems today are usually supported by what is generally known as *attack languages*. The need for attack languages can be manifold. It could be needed to understand what attack has already happened; it could be needed so that multiple intrusion detection systems can communicate with each other; or it could be required to control what attack types the IDS should detect. These languages may also be needed to reproduce attacks for testing purposes. The need for knowing about attack languages

Figure 2.4: This figure shows a birds-eye view of the interaction between the various agent types in a correlation routine

arises because of the design and implementation of an XML based Agent Deployment and Correlation Script (ADCS) in APHIDS++.

According to [19, 48] attack languages can be broken down into the following, based on what their intention of use is:

- *Event Language*: Event languages are generally used to describe a standard format for the *reporting* of *events* as they are captured. Some examples include the standard format for syslog messages, tcpdump packets, SNORT alerts in CSV format etc. These events serve as basic inputs to most security analysis tasks like performing alert correlations.
- *Response Language*: Response Languages are used to *respond* to attacks once they are detected. Most response languages, as done in APHIDS (Action Agents) use standard C, Java or Perl functions to respond to attacks. Though these types are extendable, they do have limitations on customizability.
- *Reporting Language*: These languages describe the final results of the attack correlation process in a human readable format to a system administrator. For example, the APHIDS system presents a readable message format in its user interface controlled by the UI Agent. Some of the things that a reporting language refers include the source and destination of the attack, the IDS that detected the attack, the time of the attack etc. Two of the most popular reporting languages used are IDMEF (Intrusion Detection Message Exchange Format) [14] and CISL (Common Intrusion Specification Language).

-
- *Correlation Language*: Most research on attack languages today focus on this type of language. These languages are used to *correlate* the information gathered from multiple intrusion detection systems. These languages are used to specify the relationship among attacks which could be temporally and spacially related. One of the most popular languages is STATL [19] which is an extensible state transition-based language. This language has been applied to both host-based intrusion detection and network-based intrusion detection (NetSTAT [49]).
 - *Exploit Language*: These languages are used to describe steps taken to actually perform an intrusion. These languages usually are written in standard programming languages such as C, C++, Python, etc. Examples of these languages include NASL (Nessus Attack Specification Language) [17].
 - *Detection Language*: These languages are commonly called *attack languages* [48]. These language provide the actual logistics for detecting an attack. The languages that fall into this category include the STATL language[19] and those used by the Bro system [37].

Chapter 3

APHIDS++: Policy Agent

This chapter defines the improvements that were made to the initial APHIDS design (discussed previously in section 2.4). One major improvement to the APHIDS framework was the addition of an intelligent agent, called the Policy Agent.

3.1 Need for Policy Agent

The Policy Agent is a core component of APHIDS++. It was introduced to coordinate the behavior of Trigger and Correlation Agents, which in turn would allow the system to avoid performing superfluous correlation tasks, a profusion of which may ultimately lead to denial of service attack vulnerabilities on the system itself. The Policy Agent operates by consulting a set of rules, which it modifies continually, thereby learning from previous intrusion attempts.

Our motivation for incorporating a Policy Agent in APHIDS++ follows directly from the Blaze system [41]. Blaze is a theoretical IDS designed specifically for VoIP (Voice over Internet Protocol). Contrarily, APHIDS++ is a generic IDS suitable for a variety of applications, including VoIP intrusion detection. Figure 3.1 diagrams the interaction between the Policy Agent and the original APHIDS agents, while an approximate sequence of interactions between various agents in APHIDS++ is modeled in section 3.2.

3.2 APHIDS++ Agent Interaction

In this section we describe the interaction that takes place between the agents in the new APHIDS++ design and an approximate detail of the steps that take place.

Like in the previous APHIDS system the first agents that get setup on APHIDS++ initialization are the UI Agent and the System Agent. Once they are deployed, the user runs the *enable <correlationRoutine>* command to deploy and initialize the required *trigger,policy* and *correlation* agents.

Figure 3.1: Interaction between agents with the presence of Policy Agent in APHIDS++

The *task* and *action* agents are also initialized but are not deployed. Once this setup has occurred, the following are the steps that allow for agent interaction:

1. Trigger Agent detects a probable intrusion event.
2. The Trigger Agent sends a message regarding the type of intrusion event to the Policy Agent.
3. Upon receiving a message from the Trigger Agent, the Policy Agent consults its set of rules in order to decide whether the intrusion event should be reported to the Correlation Agent. The decision is then sent, along with additional data, as a notification to the Trigger Agent.
4. If the notification from the Policy Agent contains a request to forward the intrusion event to the Correlation Agent, the Trigger Agent prepares a message, containing the data collected in step 1 as well as the data sent to it by the Policy Agent, and sends it to the Correlation Agent.
5. The Correlation Agent then spawns one or more Task Agents to gather more data regarding the reported trigger event. The data returned by Task Agents is then correlated by the Correlation Agent.
6. The Correlation Agent then queries the Policy Agent for any information pertaining to the correlated data.

7. The Policy Agent sends the requested data to the Correlation Agent.
8. The Correlation Agent, after comparing the data collected from the correlated task and the data received from the Policy Agent, may send an update request to the Policy Agent to update its rules or data. These rules are best used by the Policy Agent while responding to similar future attacks in step 3. This, in effect, facilitates the Trigger Agent's intelligent handling of identical intrusion events.
9. The Correlation Agent finally invokes an appropriate Action Agent.

3.3 Policy Agent Design Decisions

Given the importance of reduced latency in intrusion detection systems, our main goal was to introduce an intelligence into our agent framework without increasing the overhead for agent setup and communication. Hence the following are the design decisions that were taken into account while designing the Policy Agent:

1. *Semi-mobile Policy Agent*: As mentioned in section 2.1.1, in order to allow for faster response in an agent environment, with some flexibility, it is best to use semi-mobile agents. Hence to reduce latency that has been introduced by using a Policy Agent, we have opted to use a semi-mobile Policy Agent.
2. *Compatibility with previous agent system*: Given the fact that a major change has been introduced into the core APHIDS agent framework due to the introduction of a Policy Agent, we have tried to implement the Policy Agent such that it would require minimal changes to the previous agents used in the previous system. More on this is discussed in section 3.4.

3.4 Implementation Details

3.4.1 Grasshopper Environment

As mentioned in [15], the mobile agent platform used to run and deploy the APHIDS and the APHIDS++ systems is called Grasshopper [9]. This system is a java-based system that requires a Java virtual machine platform. In order to understand the deployment scenario covered in section 3.4.2, one

needs to clearly understand the logical structure of Grasshopper. The main components of Grasshopper according to [9] include:

- *region*: The region is a managerial component of grasshopper which keeps track of all the agents and their current location in the entire network. In case of APHIDS, there is one region that keeps track of all the agents.
- *agency*: In order for any agent to work on a host, there has to be atleast one agency running in the host. There can be more than one agency running in a host.
- *place*: A place is a logical division within an agency. An agency can have one or more places.
- *agent*: An agent as the name suggests, is the smallest component in grasshopper. This agent could be a static or a mobile agent. An agent has to reside in a given place.

Thus the basic structure of the environment in grasshopper is *agent* belongs to a *place*, a place belongs to an *agency*, agencies are managed by a *region*. Figure 3.2 provides the structure as mentioned in this section.

Figure 3.2: This figure shows the presence of agents deployed using Grasshopper agent environment

3.4.2 Deployment

Given the grasshopper agent environment as shown in the previous section, in order to deploy the various components (agents) of APHIDS++, all that is required is to run an agency on each host where the agents are to reside and a *place* on each of those agencies for the agents. Before any of the agencies are deployed however, we have to first deploy a *region*. Thus the following are the steps taken to deploy APHIDS++.

1. Deploy a *region* that is going to handle all the agencies and agents in Grasshopper.
2. Deploy *agencies* in all hosts where APHIDS++ agents are to travel to, and create a *place* in the agencies.
3. Start the APHIDS++ system, which in turn starts the UI and system Agents.
4. Wait for user to turn on Correlation Routines.

3.4.3 Changes to Trigger and Correlation Agent

In the previous APHIDS design, there did not exist a Policy Agent. In the new APHIDS++ design, the Trigger Agent is allowed to send queries to the Policy Agent. Hence the Trigger Agent has been modified to allow it to locate the necessary Policy Agent by contacting the *region* that maintains track of all the agents in the system.

The Correlation Agent has also been modified to allow it to track the Policy Agent. The Correlation Agent is the one who has to designate the name of the Policy Agent associated with the correlation routine. The correlation agent can query the Policy Agent similar to the Trigger Agent. It can also ask for the update of rules stored in the Policy Agent.

3.4.4 Communication with Policy Agent

The Policy Agent has been designed such that it behaves as a semi-mobile agent. The Policy Agent is created in the location where the System Agent resides. Once the user enable a routine, the Policy Agent is created and then moved to a user specified location. As mentioned in the previous section, the Trigger Agent and the Correlation Agent are the ones which communicate with the Policy Agent.

The Policy Agent only replies to requests made. It does not contact other agents on its own. The Trigger and Correlation agents have a synchronized communication proxy link to the Policy Agent i.e. they have a synchronized communication channel to the Policy Agent. This was done to reduce the latency incurred in moving agents around in the system as mentioned previously.

3.5 Evaluation of Policy Agent

In order to evaluate the effectiveness of the introduced policy agent in APHIDS++, we perform two different sets of tests.

The first test performs a DoS attack against the APHIDS++ and the APHIDS systems. We then evaluate the effectiveness of the Policy Agent by comparing the results obtained from the base APHIDS systems and the new APHIDS++ system.

The second test gives a description of the evaluation of APHIDS++ with respect to detection of a VoIP attack on a H.248 system by applying a well known misuse detection technique to user profiles.

3.5.1 DoS attack on APHIDS++

The purpose of this test is to compare the effectiveness of the new and improved APHIDS++ systems using the Policy Agent, compared to the previous APHIDS system.

3.5.1.1 Test Environment Setup

In order to allow for the best comparison to results obtained by Deeter in [15], we have used the same equipment setup as used previously. Table 3.1 provides the configuration of PCs used for testing purposes. The interconnectivity between the PCs and the location of each of the agent types is shown in Figure 3.3.

3.5.1.2 Test Methodology

In [15], the author had conducted a test where the APHIDS system was bombarded with numerous portscan operations from the same source to the same target. As a result, the system showed exponential rise in computational time (latency) to correlate all the data. This was in effect, a DoS attack on the APHIDS system. We had previously experimentally shown a

Machines Used	PC Configuration
Aphids Console	1.0 GHz AMD Athlon; 256MB RAM
SNORT system	200MHz Pentium-Pro; 160MB RAM
Attacker	200MHz Pentium. 96MB RAM
SSH Server	200MHz Pentium-Pro. 160MB RAM
Web Server	Dual 200MHz Pentium-Pro. 160MB RAM
Hub	3com LinkBuilder 10BTi

Table 3.1: This table shows the configuration of the hardware used to compare the DDoS test performed on APHIDS++ and APHIDS

three fold reduction in latency and bandwidth consumed using *agent caching* to cache *Task Agents* in APHIDS++ [1]. In this section we provide a test that does not use agent caching.

In APHIDS++, the Policy Agent can be used to detect and stop unnecessary correlation tasks early on. In order to prove the effectiveness of the Policy Agent, we perform the same task of performing continuous portscan operations. As has been explained previously, the Policy Agent is a *learning* system that learns from previous correlation attempts of the same type of intrusion. Hence if an intruder bombards the system with numerous portscan operations, and if this is the first time the intruder has performed this action, the Policy Agent will not be effective, as it has not learnt about the attack yet. Hence in order to allow for the Policy Agent to learn, the following steps are followed for the test:

1. User does one portscan from attacker IP to target IP using the "nmap" command, which is the command used for the portscan operation.
2. This portscan operation is detected by the SNORT system and written into its logs.
3. This log is read by the Trigger Agent, which detects the intrusion.
4. The Trigger Agent then contacts the Policy Agent with the source and destination IP of the attack.
5. Since this is the first time that the attack has taken place from a source and a target, the Policy Agent notifies the Trigger Agent to continue reporting the task to the Correlation Agent as the Policy Agent has no prior knowledge about this attacker. Do note that the

Figure 3.3: This figure presents the setup used to compare APHIDS and APHIDS++ latency measurements

Policy Agent could insert the Source and Destination IP of the attack into its database, which would have stopped any further portscan correlations. But we are trying to prove the effectiveness of the Policy Agent as a learner and not as an initial blocker. This blocking operation could have been done at the Trigger Agent itself, but that is not the purpose of this test.

6. The Trigger Agent, after receiving a clearance from the Policy Agent, reports the attack to the Correlation Agent.
7. The Correlation Agent in turn launches task agents to gather SSH log files from the SSH server.
8. The Task Agents travel to the SSH server, and reports the data back to the Correlation Agent.
9. The Correlation Agent then performs the correlation task, and reports the source and destination IP of the attack to the Policy Agent so that it can block intrusions which originate from the same attacker to the same target. Do note however that the Policy Agent does not block

portscans which have the same source IP of the attack but a different target.

10. Now that the information about the attack has been stored by the Policy Agent, it will block future attacks from the same source and the target for a given time window. This is a re-adjustable window that could vary according to the number of attacks from a given source. In our case, we have set it to block the attacker indefinitely from the time that the Correlation Agent had reported the event to the Policy Agent.
11. Now the user performs multiple portscans using a script. For every nmap, SNORT reports three portscan operations. For every portscan, the Trigger Agent receives one trigger event.
12. The Trigger Agent then contacts the Policy Agent for each Trigger Event with the source and destination of the attack.
13. The Policy Agent matches the information received with its database of previous attacks that it has been asked to monitor.
14. The Policy Agent detects that this attack should not be correlated. Hence it notifies the Trigger Agent. The Trigger Agent after receiving this notification, drops the Trigger Event and does not forward it to the Correlation Agent.

3.5.1.3 Test Results

As shown in Figure 3.4, we were able to considerably reduce the system latency introduced in APHIDS as a result of unnecessary correlation tasks, for example, previously it took almost a minute to complete the correlation process for 60 portscans, now it takes less than a minute. The raw test data used for the figure is shown in table 3.2. Though the latency period reduces, the relative growth rate in latency period in both cases keep increasing equally, as shown in the log scale in Figure 3.4.

For APHIDS, the latency period was calculated by measuring the time taken from the point when the Correlation Agent is invoked by the Trigger Agent to the time when the Action Agent is deployed.

For APHIDS++, the latency period was calculated at the Policy Agent by measuring the time taken at the point when the Trigger Agent contacts the Policy Agent and invokes its set of rules. The difference between the

Figure 3.4: This figure shows a log scale comparison of the latency measurements obtained from portscan tests in APHIDS++ and APHIDS

Portscans	APHIDS (millisec)	APHIDS++ (millisec)
5	3641	16
10	10453	31
20	19922	93
30	30016	156
60	59609	266

Table 3.2: This table shows the latency period obtained from tests based on the number of portscans done on APHIDS and APHIDS++

time at first contact and time for last contact for n number of portscans gives the required time needed to calculate the computation period.

3.5.1.4 Policy Agent introducing more latency

Even though the Policy Agent drastically reduces the amount of latency, there are situations when it fails to improve the performance of APHIDS++. Infact it could introduce more latency.

As we know, there is an extra communication that has to be made in APHIDS++ compared to APHIDS. In APHIDS++, the Trigger agent has to ask for the set of rules from the Policy Agent. The Policy Agent has to process the request and then answer back to the Trigger Agent. If the Policy Agent asks the Trigger Agent to continue the correlation task, this period of time that has passed would introduce a latency. This situation is going to occur frequently in the beginning, when the Policy Agent has not learnt anything yet, or does not have the knowhow to stop unnecessary correlation tasks.

3.5.2 Detecting VoIP attack on H.248/Megaco system

In this section we test the applicability of our Policy Agent to a H.248-based VoIP scenario. As mentioned earlier, our motivation for incorporating a Policy Agent in APHIDS++ followed directly from the Blaze system [41]. Since Blaze was a proposed VoIP only design, we decided to test the Policy Agent on a *emulated* VoIP environment by making use of a misuse-based detection principle applied to user profiles.

3.5.2.1 User Profiling

Currently there does not exist any work on user profiling of VoIP subscribers. User profiling is required for detecting intrusions of VoIP user accounts or the VoIP network. Previous related work to user profiling that is available in literature include profiling user behavior using a host-based intrusion detection technique on computer usage. Variables that are measured include the frequency of softwares accessed and commands executed by the user [3]. Yuilli et al.[51] have proposed the use of *attacker profiling* to track the behaviour of attackers and to detect the resources available to them using principles of economics. Seleznyov et al. [40] propose building an anomaly detection system based on *Allens Temporal Algebra* for profiling user behavior.

Advantages of User profiling in VoIP

The advantages to user profiling are manifold. Some of the notable advantages are:

- *Detect intrusive behavior by a user:* In most cases, hackers are not new users, but regular users of the VoIP network. Given the availability of user profiles, it would be easier to detect if an existing user hacks into other users VoIP account by using both temporal (information based on time) and spatial (information gathered from multiple hosts) correlation techniques.
- *Detect a compromised user account:* Related to the previous point, a substantial change will be detected in the call behavior of a compromised user account. The existence of a user profile of the compromised account can easily detect the change.

A combination of both detections could reveal both the attacker and target if enough information is available.

Challenges to user profiling in VoIP

In building a misuse or anomaly detection-based user profiler, we have identified the following as a list of challenges that have to be overcome:

- *Use of multiple accounts/ multiple protocols:* Given the fact that there exists multiple VoIP protocols (SIP, H.323, H.248/megaco, proprietary protocols), it will be a challenge to retrieve information about users in cases where a user uses multiple user accounts and/or multiple protocols for each of the accounts. This will be a challenge as the amount of trust that can be placed on the variables gathered from a particular users' usage of VoIP will not have the full set of calls made by the user. This can be overcome only if data is shared between the user's multiple accounts.
- *Ambiguous phone usage:* Another factor that can be a major hurdle to profiling users is the lack of coherence of user calls. This is a possibility if a VoIP phone is used commercially as a public phone booth. In this case the callee will in most cases be different, and most of the variables used for measuring user profiles will also be different as almost all users making the call will be unique.

3.5.2.2 Test Environment Setup for detecting VoIP Call Hijack Attack

In order to test the effectiveness of our Policy Agent in detecting a VoIP intrusion, we emulate a VoIP H.248 network setup in a single host.

Equipment Used

We used a 3.0 GHz Intel Pentium 4 machine with 512 MB of RAM running Windows XP Professional. This host runs an Apache 2 Webserver meant for hosting the APHIDS++ class files. The PC also has a copy of Grasshopper 2.2.4 software on it.

Environment Setup

The following are the steps taken to setup the environment to simulate VoIP Reinvite Attack:

1. The first step is to start a Grasshopper Region that will control all the agencies in this host. This was done using the command

```
Grasshopper r -n shahedregion -tui.
```

2. The second step is to launch three agencies where the agents will travel to. We need three agencies. One for the Trigger Agent to simulate the Trigger Agent being placed at the Media Gateway. One for the Task Agent, to simulate its travel to the Media Gateway Controller to gather data. And finally one for the Policy Agent, to simulate its presence in a separate host. All these agencies have to register with the *region* launched in the previous step during startup. This is achieved by using the command

```
Grasshopper a -n agency1 -r socket://192.168.1.101:7020/  
shahedregion -pl aphids -tui
```

for agency 1 and so on.

3. The third and final step is to launch an agency that is meant to host the other agents. In order to launch APHIDS we have to use the following command

```
Grasshopper a -n agency4 -r socket://192.168.1.101:7020/  
shahedregion -pl aphids -tui -a aphids.LauncherAgent@aphids  
http://192.168.1.101/aphids
```

3.5.2.3 Test Methodology for detecting VoIP Call HIjack Attack

Figure 3.5: Use of APHIDS++ in a VoIP call hijack attack scenario

We have successfully used the model presented in Section 3.2 to detect a simulated VoIP Megaco/H.248 reinvite message intrusion[41], also known as *call hijacking*. In this type of attack, the attacker takes over a conversation between user-x and user-y by masquerading as user-x by sending a *reinvite* message. Here we try to explain how this misuse-based intrusion detection model works for detecting the call hijack and reduce the number of false positives by making informed judgement using user profiles.

Initially the system is configured with the Trigger Agent placed at a media gateway (MG), as shown in Figure 3.5. The Correlation Agent and the Policy Agent are setup on two separate hosts. It is possible for an attacker to hijack the current call between user-x and user-y by using the user-id of user-x to send a reinvite message to user-y through the media gateway (MG). User-x now hears only static, and will not be notified of the

call hijack in the current protocol. Moreover, he will not be able to make any subsequent calls unless he completely resets his VoIP system. If the user does not reset his system, he will continue sending *orphaned* packets (packets that will not be forwarded by the MG) meant for user-y. This acts as a probable intrusion event in our system, as mentioned in step 1 in section 3.2.

Once the Trigger Agent, sitting at the MG, receives an orphaned packet, it sends a request to the Policy Agent enquiring if the source IP of the orphaned packet (user-x) is authorized to connect to the current MG. If the Policy Agent gives a negative feedback, then the Trigger Agent does not report a trigger event to the Correlation Agent. This has the advantage of blocking distributed denial of service attacks from hosts not authorized to connect to an MG.

On the other hand, if the Policy Agent determines that the source is valid, it sends a list of IP addresses maintained in user-x's profile⁵. This list includes all known IP addresses which have been previously used by user-x in making calls.

The Trigger Agent then inspects the current calls being routed through the MG, and tries to find a match to the IP list received from the Policy Agent. A matching IP address indicates that the potential attacker IP is actually a phone in use by user-x and that the source of orphaned packets is another instrument which user-x forgot to hang up. This mechanism for detecting a false alarm avoids the unnecessary invocation of Correlation Agent and subsequent Task and Action Agents.

On the other hand, if no matching IP address was found, the Trigger Agent reports a probable intrusion event to the Correlation Agent. The Correlation Agent, in turn, launches a Task Agent in order to query the MGC for more information regarding the reinvoke message. Once the Correlation Agent receives a response from the Task Agent, it prepares a correlated user profile by correlating the information received from the Trigger Agent and the Task Agent.

After correlation, it queries the Policy Agent to obtain the entire profile of user-x and then compares it with the new correlated user profile. A large hamming distance between the two profiles indicates an indubitable intrusion event, whereas a hamming distance below a user-configurable threshold denotes a false alarm. In the event of a false alarm, the Correlation Agent

⁵A user's profile is based on his past usage of the VoIP system and summarizes attributes such as the IP addresses of phones belonging to the user, usual times of day when the user makes a call, average duration of calls, number of calls made over a period of time, etc.

adds the spurious attacker IP to user-x's profile and sends it as an update to the Policy Agent. Subsequent uses of that IP are intelligently detected by the Trigger Agent as non-intrusive behavior. Appendix B shows the visual feedback received during the simulation.

Use of User Profiles

As mentioned in the test methodology section, we measure the hamming distance between the available user profile in the Policy Agent with the profile obtained from the current call before being certain of an attack. The variables that are available in the user profile include: the location of the user, the user mode (administrative account or customer account), normal time of the day when the user makes calls, list of IPs used by the user, list of users called by the user etc. Not all profile variables are useful in detecting the attack on a user account.

In the emulated case described in this section, we throw an alarm only if the two variables: (1) location of the user sending the reinvite message, and (2) time of day when the reinvite message occurred, do not appear in the user's profile obtained from the Policy Agent. If the variable obtained from the current call do match the variables from the users profile, then the source IP of the reinvite message is added to the user's profile to avoid false positives.

Chapter 4

APHIDS++: ADCS

The first proposal for using a correlation script for APHIDS was made in [16]. Agent Deployment and Correlation Script (ADCS), as it is called in our APHIDS++ system, is a high-level specification language used to describe an intrusion detection correlation routine.

An ADCS is required not only for describing and initializing the agents involved in a correlation routine, but also for defining the dynamic state of a correlation routine during runtime. This dynamic state could be as simple as printing a message to notify the system administrator of what cases have failed, or as complicated as launching different types of mobile agents that gather data from other hosts, for example, which Action Agent to invoke, or when to launch more Task Agents etc. The dynamic state is achieved through various test conditions which are part of every routine. We label this set of conditions as the *Rule Set* of every correlation routine.

4.1 ADCS categorization

Related to the categorization of IDS languages as mentioned in section 2.5, our ADCS can be categorized as a *detection* and a *correlation* language. We consider the language as a detection language as this language uses rules for each correlation routine that are evaluated to detect various conditions of an attack. We also consider the language as a correlation language, as the rules available to each routine is evaluated by testing conditions based on results gathered from multiple IDSs.

As our current version is an initial draft of our ADCS, the current version of our ADCS has low functionality with regards to what can be achieved. But the future versions of our ADCS will be able to behave as a *response* language as well by launching Action Agents in response to attack detections.

4.2 Design decision to use XML

In APHIDS++, we have used XML as the scripting language for our ADCS. There were many reasons for using XML in our system rather than building our own scripting language from scratch. Some of them can be outlined as:

- First, the availability of a built in validation scheme in XML scripts using DTD (Descriptive Type Definition) avoids complications that could develop due to incorrect ADCS file input from a user. This allows us to create a DTD specifically for APHIDS++ that can match the pattern of the input ADCS file according to the DTD rules.
- Second, the availability of XML visualization and construction tools such as the commercially available system, XMLSpy ⁶ facilitates the management of large quantities of scripts. There are literally over a hundred different intrusion detection systems and each of them detect a multitude of attack types. This increases the probability of having thousands of correlation scripts at any point in the future. Hence the need for XML visualizers to track the deployment state of each script is necessary.
- Another reason for choosing an XML scripting approach was due to the current inclination of IDWG (Intrusion Detection Work Group) of IETF (Internet Engineering Task Force) to use an XML based approach for IDMEF (Intrusion Detection Message Exchange Format) [14] ⁷. The inclination to use XML by a standardization body for intrusion detection message exchange would allow our system to be compatible with the IDMEF standard in the future. Though the IDMEF standards do not have mobile agent behavior included, they do have XML tags for proprietary information that allows for individualized design of scripts.
- Finally, we believe that XML's intrinsic extensibility will be highly beneficial to our system in the long term due to the fact that we present here only an initial prototype of our system that has limited functionality. The extensible nature of XML would allow our ADCS to grow in the future to allow for more data types and added statistical functionality that is needed to test conditions in some anomaly-based

⁶available at <http://www.altova.com/>

⁷IDMEF is used as a common language for communication between different intrusion detection systems.

detection techniques, that need to learn based on previous intrusion attempts.

4.3 XML related work in mobile agents

Though we have discussed the use of attack languages, our work also correlates to use of XML with mobile agents in the mobile agents community. Using XML with mobile agents is not a new topic. It has been used in the XMARS system [11] for communication between mobile agents. In the XMARS system, agents communicate with each other using tuples of data. A tuple is simply a list of values that are matched. An agent interacts by waiting on a specific set of tuples which are provided by other agents. A similar work has also been proposed by Lu et al. in [31] where they apply the tuple-space to their system called XSpace.

4.4 ADCS Implementation

In this section we provide the details about the libraries, and data structures used to implement our ADCS in XML.

4.4.1 XML Parsing Basics

Presently, two major API specifications define how XML parsers work: SAX (Simple API for XML) and DOM (Document Object Model).

DOM defines a tree-based approach to navigating an XML document i.e. a DOM parser processes XML data and creates a object-oriented hierarchical representation of the document that can be navigated as required during program execution.

SAX on the other hand is an event-based approach i.e. whenever a certain type of tag is encountered while processing an XML file, it calls handler functions meant for each type of event (for example, begin document, begin element, begin attribute etc.)

The advantage of DOM compared to SAX is the ease with which documents can be parsed by using the *parent* or *child* command on nodes of the hierarchical tree built by DOM. DOM is usually used when the DOM tree has to be manipulated at run-time.

The disadvantage of DOM compared to SAX is that due to the internal representation of a DOM tree, parsers that use DOM need to have large memory capacities. DOM trees usually create nodes for even the most minor

parsing characters, for example there are nodes for each new line character present in an XML document.

SAX on the other hand can handle gigabytes of XML data without using too much of the system resources as it does not maintain such a tree. However, if a developer decides to use SAX, he would have to create his own parsing functions and own parsing tree to maintain the data obtained from reading the XML file.

4.4.2 Xerces XML for java library

In order to allow our system to use XML, we have used an open source XML parsing library, Xerces (available at <http://xml.apache.org/xerces-j/>), for implementing our script parser. We have used Xerces version 2.7.1 as the linked library.

Xerces is an open source version used for parsing XML using java. Given the fact that our APHIDS++ system is a java-based mobile agent system, it was easier for us to integrate our system with a java-based library. Xerces provides parsing capabilities in both DOM and SAX.

In our APHIDS++ system, we use SAX as our parsing model. The reasons for choosing SAX over DOM is because in the long run, we believe that we will have thousands of correlation scripts that handle different types of intrusions which would then require a lot of system resources that we may or may not be able to support.

As mentioned in the previous section, the disadvantage of SAX is the requirement to create a personalized data tree to allow reusing data. In APHIDS++ we maintain a tree of Hash Maps for every correlation routine. The tree structure for a correlation routine is shown in Figure 4.2 and Figure 4.3. More on the description of the tags saved in this tree is explained in the subsequent sections.

4.4.3 Description of XML Syntax and rules

This section describes some of the syntax used in defining a correlation routine using XML in APHIDS++. Figure 4.1 shows the syntax of our correlation routine in the XML format. The next few headings define each of the elements used to construct our XML script. The reader should refer to Figure 4.1 to understand the nested structure of the tags used in our XML ADCS.

```

<?xml version="1.0" encoding="UTF-8"?>
<allRoutines>
<correlationRoutine class="aphids.correlation.test.TestCorrelation" identifier="aphids1"
codebase="http://192.168.1.101/aphids/" deploy="false">

  <trigger class="aphids.trigger.prototype.ShortPortScanTrigger" identifier="a" arg="1">
  </trigger>
  <task class="aphids.task.prototype.FindUsersFromHost" identifier="b" arg="1,2">
  </task>
  <policy class="" identifier="" arg="">
  </policy>
  <action class="aphids.action.prototype.ConsoleMessageAction" identifier="c" arg="">
  </action>
  <ruleSet>
    <rule name="rule1" condition="a.source_ip == 192.168.1.20">
      <true message="This is a known attacker.">
      </true>
      <false message="New attacker."></false>
    </rule>
    <rule name="rule2" condition="b.name == john">
      <true message="task stop">
      </true>
      <false message="task continue"></false>
    </rule>
  </ruleSet>
  <otherData>
  </otherData>
</correlationRoutine>
</allRoutines>

```

Figure 4.1: XML code showing the description of a correlation routine

tag: allRoutines

The upper most tag used is *allRoutines*. This is used to encapsulate all the correlation routines in our system. This is more of an XML requirement used to construct the root of an XML tree. Hence it is not stored in our hash table data structure as shown in Figure 4.2 and Figure 4.3. Figure 4.4 shows the usage of the *allRoutines* tag to encapsulate all the correlation routines.

tag: correlationRoutine

The upper most relevant tag used is the *correlationRoutine* tag shown in Figure 4.5. This block contains the description of a correlation routine to

Figure 4.2: This figure shows the structure of the HashMap generated for a correlation routine after reading the XML-based ADCS

be performed. Some of the attributes that are used with this tag include:

- *class*: This is probably the most important tag in the XML file. This attribute is used to signify which correlation routines are available in the system. This variable is also used to start and stop correlation routines using the UI Agent.
- *identifier*: An identifier is used to describe data collected by this correlation routine. This identifier can be used in the future to report data back to a system administrator if APHIDS++ ADCS is used as a *reporting* language.
- *codebase*: This variable gives the location where all the source files for this routine are available.
- *deploy*: This variable signifies the default deploy condition (to automatically start the routine upon system initialization).

Figure 4.3: This figure shows the structure of the HashMap generated for the ruleSet variable of a correlation routine

```

<allRoutines>
  <correlationRoutine class="x" ...>
  ...
</correlationRoutine>
  <correlationRoutine class="y" ...>
  ...
</correlationRoutine>
</allRoutines>
  
```

Figure 4.4: This shows the usage of the allRoutines tag to encapsulate all the correlation routines in ADCS

tag: trigger

This tag is used to designate the variables that control the deployment of the Trigger Agent. The following are the attributes used:

- *class*: This attribute is used to designate the class name of the Trigger Agent of the correlation routine. This name has to match the actual class name that is to be used.
- *identifier*: This attribute is used to mark data that is sent by this Trigger Agent to the Correlation Agent. The data values can then be used by *rules* for comparison purposes.
- *arg*: List of arguments used to initialize the Trigger Agent.

```
<correlationRoutine class="<java class>" identifier="<x>" codebase="<web address>" deploy="<true|false>">
...
</correlationRoutine>
```

Figure 4.5: This shows the usage of the correlationRoutine tag with its list of attributes

tag: task

This tag is used to designate the variables that control the deployment of the Task Agent. The attributes used is the same as the ones used by the Trigger Agent.

tag: policy

This tag is used to designate the variables that control the deployment of the Policy Agent. The attributes used is the same as the ones used by the Trigger Agent.

tag: action

This tag is used to designate the variables that control the initialization of the Action Agent. The attributes used is the same as the ones used by the Trigger Agent.

tag: ruleset

This tag is used as a container to contain the various *rules* that will be checked by the correlation routine during the correlation process. Figure 4.6 shows an example of its usage. Figure 4.3 shows the Hash Map that is maintained for each rule set.

tag: rule

This tag is the most complicated label that actually gives dynamic results based on the correlation process. The two attributes used by the rule tag are:

- *name*: The name of this rule, so that it can be distinguished from other rules in this correlation routine.

```
<ruleSet>

  <rule name="<x>" ...>
  </rule>

  <rule name="<y>" ...>
  </rule>

  <rule name="<z>" ...>
  </rule>

</ruleSet>
```

Figure 4.6: This shows the usage of the ruleSet tag used to encapsulate the set of rules of a correlation routine

- *condition*: The dynamic condition that is to be checked during run-time. The current prototype just has a (lhs op rhs) syntax which is used to check the rule. More on conditions is explained in a later section.

Figure 4.7 shows the structure of the rule tag with the nested elements present in it.

tag: true

This tag is executed only if the condition evaluated by the rule tag is true. If it is, the current version of the XML ADCS just outputs a message which is printed on the UI by the UI Agent. In the future this message could contain variables which are parsed to deploy additional Task or Action Agents. Or this tag could have more rule tags allowing for a greater depth in evaluations as is available in most scripting languages. For now we consider this as our future work.

tag: false

This tag is executed only if the condition evaluated by the rule tag is false. Currently its functionality is the same as the *true* tag.

```
<rule name="rule1" condition="a.source_ip == 192.168.1.20">
  <true message="This is a known attacker."> </true>
  <false message="New attacker."></false>
</rule>

<rule name="rule 2" condition="b.numberOfLogins gt 5">
  <true message="Regular user"></true>
  <false message="unknown user"></false>
</rule>
```

Figure 4.7: This shows the usage of the rule tag with its set of attributes and nested elements

tag: otherData

This tag is not currently used to do anything. This tag has been placed to allow for the addition of future variables to the XML script without change in the major tags already described.

4.4.3.1 Condition Evaluation

Given the current infancy of APHIDS++ ADCS, there are currently limitations to what set of conditions can be evaluated in each *rule* tag.

- *lhs op rhs*: The current condition check has to have a tri-variable condition separated by spaces. The current condition parser cannot parse nested condition structures.
- *Operators accepted*: As mentioned in the previous point, the second argument of the condition is the operator. The list of current evaluated operators include ==, !=, gt and lt where *gt* and *lt* are used instead of < and >. This is because < and > are reserved characters in XML.
- *evaluated values*: The current system can evaluate variables that arrive only from the Trigger, Task and Policy Agents. The Action Agent is invoked only after the correlation process. Hence it is not evaluated.

4.5 Evaluation of XML-based ADCS

In order to test the effectiveness of the XML-based ADCS, the following were the tests conducted and the resulting behavior.

4.5.1 Multiple correlation routines

Figure 4.8: This shows the use of the 'list' command for checking that multiple correlation routines were successfully read from the ADCS

To test the effectiveness of the system in handling multiple correlation routines, we added two fully formed, valid correlation routines into our XML file. We then initialized APHIDS++ by starting the system. Previously, all the correlation routines were listed in a routines.txt file. We disabled the call to this file and checked if our XML file was read properly.

In order to check the behavior, the user runs the *list* command from the UI Agent. On applying the command, the user gets the name of the correlation routines available only in the XML file. This test successfully passed in our system. Figure 4.8 shows the test result.

4.5.2 Proper invocation of Policy Agent

Figure 4.9: A message is displayed to the user by the Trigger Agent after it gets data from the Policy Agent showing proper Policy Agent was invoked

In our intermediate APHIDS++ system, the user had to explicitly mention the Policy Agent to be used using the `deployPolicy()` method in the user instantiated `correlationObject` class. In order to test the functionality of the Policy Agent tag, we removed the `deployPolicy()` method from the APHIDS++ system and tried invoking the correlation routine with the XML file.

The test result showed the deployment of the correct Policy Agent without problems as the correlation routine completed its task without problems. Figure 4.9 shows that the correct Policy Agent was invoked by the script.

4.5.3 Proper invocation of Trigger Agent

In the base APHIDS system, the user had to explicitly mention the Trigger Agent to be used using the `deployTrigger()` method in the user instantiated `correlationObject` class. We removed this method from the system and tried our test by starting a correlation routine using the ADCS.

The test results showed that the correct Trigger Agent class was deployed using the parameters passed using the XML file. Figure 4.10 shows the Trigger Agent of the VoIP call hijack emulation routine calling the Policy Agent.

Figure 4.10: Test to show the Trigger Agent was invoked properly from within the script: The Trigger Agent is contacting the Policy Agent to check if the sourceIP of the orphaned packet is in the allowed list of IPs that can use this MG

```
2 1 on here
The rule info that we have is rule2 :b.name == john
Value of lhs2 is null
result is false
The rule info that we have is rule1 :a.source_ip == 192.168.1.20
Value of lhs2 is 192.168.1.20
result is true
```

Figure 4.11: Test to show the proper invocation and evaluation of rules

4.5.4 Proper Evaluation of Rules

The most important component of our XML file is the use of rules which get evaluated during runtime. In order to test the effectiveness of our rule parsing and the use of identifiers, we run a test such that the value returned by the Trigger Agent is tested to see if the source IP of the attacker matches a previous attacker IP address. This test is done using a test condition: `(a.sourceip == 192.168.1.20)` where *a* is the identifier of results obtained from the Trigger Agent. And *sourceip* is the source of the attack.

We then ran an attack which was detected by SNORT with an attack source ip of 192.168.1.20. By default, our condition test returns false. But in this case, it returned a true value. Figure 4.11 shows that our rule evaluates properly where the first rule is false due to the non-existence of a variable *b.name* and the second rule is true because of the proper match. Another set of test results are shown in Figure 4.12 with multiple rules.

Figure 4.12: Test to show the proper evaluation of rules as seen in the APHIDS++ UI

Chapter 5

Conclusion and Future Work

This masters thesis explains the enhancements made to the initial design of a mobile agent-based hybrid intrusion detection system [15], which correlates information gathered from multiple intrusion detection systems.

This thesis provides two main contributions. The first is a detailed description about the improvements made to the initial APHIDS framework in the form of an intelligent Policy Agent. This Policy Agent applies a feedback mechanism between the Trigger and the Correlation Agent, allowing the system to learn from previous intrusions of a similar type.

The effectiveness of the Policy Agent has been shown by two tests. One is a direct comparison to the test done in [15] where a Denial of Service attack is attempted on the system through multiple portscan operations. A comparison between the latency obtained in the earlier system is compared with the new enhanced system, showing the improved intelligence of the APHIDS++ system compared to its earlier version in avoiding processing of unwanted intrusion alerts.

The second test simulates and shows how the system can be deployed in a VoIP (H.248/ Megaco) environment. We apply simple misuse-based detection techniques on the learnt user profile of users to gather probable intrusion alarms.

We also proposed and implemented an XML-based correlation language that allows for system initialization and rules that may be further extended to allow for easy processing and control of the entire APHIDS system. The reason for using XML was for the reason that it is easy to extend, without too many changes to the initial system design. We have allowed for a *otherData* tag to allow extensions in the future.

Though this thesis has fixed many of the flaws that were found in the initial system design, there are other improvements that can be made to the system to allow for better performance of the system in general. The proposed future improvements to the system can be broken down into three sections, namely, improvements to APHIDS++ in general, improvements to the Policy Agent and lastly improvements to the XML-based ADCS. We now enlist the improvement under each of the mentioned categories.

5.1 Improvements to APHIDS++

- The current APHIDS design has just a single Trigger Agent that detects probable intrusion alerts. A better system could be developed such that one Trigger Agent can maintain a state machine within it and report intrusions only when the system reaches an end state. Though such features are explicitly built into the SNORT system used in testing APHIDS, it would be of research significance if the number of intrusions reported to the Correlation Agent could be reduced, to lessen the latency incurred in correlation tasks.

Alternatively, one could use multiple Trigger Agents sitting on top of different hosts, connected to the Policy Agent. The Policy Agent could maintain the required state machine. The Policy Agent could inform the Trigger Agents to report an alarm to the correlation agent if and when an end state is reached that requires reporting intrusions.

- Currently, the bottleneck in the APHIDS++ framework is the Correlation Agent. One possible avenue to reduce latency would be to allow the Trigger Agent to randomly choose one of several copies of Correlation Agents available, that further correlates tasks. This would be important for two reasons. One is reduced latency as the load would be evenly balanced among the multiple Correlation Agents; the other is the effectiveness of this method in avoiding a central point of failure by distributing the workload among multiple Correlation Agents in multiple hosts.
- In APHIDS++, we have emulated a VoIP attack and deduced emulation results. It would be wise if the APHIDS++ system was deployed in an actual VoIP environment to test its effectiveness in detecting VoIP intrusions.
- Currently we do not apply any encryption in the agent communication. Once the system has been further developed, it would be vital to have encrypted communication between the agents, as the consequences would be disastrous if the agents themselves cannot be trusted when they are meant to guard the network.
- The current APHIDS++ system does not monitor if all the agents that are supposed to be running at different hosts, are actually alive. It would be best if some mechanism is placed within the APHIDS++

framework to periodically test if all agents are alive and well in the system.

- Once the system has been developed with at least the above considerations, it would be best if the APHIDS++ system is deployed in larger environments to test its effectiveness in handling real-time workloads. Current real-time IDSs have to process information in milliseconds on average. APHIDS++ is a long way away from it still. With the improvement of Java Virtual Machine (JVM) technology, the latency cost to deploy such a system is decreasing.

5.2 Improvements to Policy Agent

- Though the APHIDS++ system has incorporated an intelligent Policy Agent into the APHIDS framework, the true potential of its use cannot be obtained until and unless better learning avenues have been applied, tested and compared. This is needed as there are other learning methods based on machine learning principles, neural networks, bayesian networks and other stochastic local search techniques from the area of bioinformatics, which can be used within the system. One could research into applying these techniques to obtain the best system that reduces the work load of the APHIDS++ system and reduces the number of false positives obtained.
- In this thesis, we have only used misuse-based detection of a simulated VoIP network. A future possibility could be to detect ambiguous usage of a VoIP network using anomaly detection techniques.

5.3 Improvements to Agent Deployment and Correlation Scripts

- The use of XML based correlation scripts in this thesis is just a first attempt to build a correlation attack language geared towards APHIDS++. It would be of importance to further develop the language such that more logistics can be put into the XML based language. This would be useful as the more the language grows, the easier it will be for administrators to reuse already programmed components in the APHIDS framework thus reducing workloads in the long run.

-
- Currently we do not invoke agents from within a rule. In the future such modifications should be allowed and tested.
 - The current APHIDS++ ADCS checks for only a three parameter condition in a rule. In the future one could try multi-level condition check using tools such as JEP - Java Math Expression Parser if the intention is to use an anomaly detection condition in a rule from within ADCS.

Bibliography

- [1] Mohammed S. Alam, Abhishek Gupta, Jake Wires, and Son Vuong. Aphids++: Evolution of a programmable hybrid intrusion detection system. In *Second International Workshop on Mobility Aware Technologies and Applications*, 2005.
- [2] Yi an Huang and Wenke Lee. A cooperative intrusion detection system for ad hoc networks. In *SASN '03: Proceedings of the 1st ACM workshop on Security of ad hoc and sensor networks*, pages 135–147, New York, NY, USA, 2003. ACM Press.
- [3] Debra Anderson, Thane Frivold, Ann Tamaru, and Alfonso Valdes. Next-generation intrusion detection expert system (nides), software users manual, beta-update release. Technical Report SRI-CSL-95-07, Computer Science Laboratory, SRI International, 333 Ravenswood Avenue, Menlo Park, CA 94025-3493, May 1994.
- [4] Spyros Antonatos, Kostas G. Anagnostakis, and Evangelos P. Markatos. Generating realistic workloads for network intrusion detection systems. In *Proceedings of the fourth international workshop on Software and performance*, pages 207–215. ACM Press, 2004.
- [5] Midori Asaka, Atsushi Taguchi, and Shigeki Goto. The implementation of ida: An intrusion detection agent system. In *Proceedings of the 11th FIRST Conference*, 1999.
- [6] Jay Aslam, Marco Cremonini, David Kotz, and Daniela Rus. Using mobile agents for analyzing intrusion in computer networks. In *Proceedings of the Workshop on Mobile Object Systems at ECOOP 2001*, July 2001.
- [7] J. S. Balasubramaniyan, J. O. Garcia-Fernandez, D. Isacoff, E. Spafford, and D. Zamboni. An architecture for intrusion detection using autonomous agents. In *ACSAC '98: Proceedings of the 14th Annual Computer Security Applications Conference*, page 13, Washington, DC, USA, 1998. IEEE Computer Society.

-
- [8] P. Barford and D. Plonka. Characteristics of network traffic flow anomalies, 2001.
 - [9] C. Baumer, M Breugst, M Choy, and T Magedanz. Grasshopper – a universal agent platform based on omg masif and fipa standards. In *First International Workshop on Mobile Agents for Telecommunication Applications (MATA '99)*, pages 1–18, 10 1999.
 - [10] A. Bieszczad, T. White, and B. Pagurek. Mobile agents for network management. *IEEE Communications Surveys*, 1998.
 - [11] Giacomo Cabri, Letizia Leonardi, and Franco Zambonelli. XML datas-paces for mobile agent coordination. In *SAC (1)*, pages 181–188, 2000.
 - [12] Suresh N. Chari and Pau-Chen Cheng. Bluebox: A policy-driven, host-based intrusion detection system. *ACM Trans. Inf. Syst. Secur.*, 6(2):173–200, 2003.
 - [13] M. Crosbie and G. Spafford. Defending a computer system using au-tonomous agents. In *8th National Information Systems Security Con-ference*, 1995.
 - [14] H. Debar, D. Curry, and B. Feinstein. The intrusion detection message exchange format, 2005.
 - [15] Ken Deeter. Aphids: A mobile agent-based programmable hybrid in-trusion detection system. In *Masters thesis, Department of Computer Science, University of British Columbia*, 2004.
 - [16] Ken Deeter, Son Vuong, Kapil Singh, Steve Wilson, and Luca Filipozzi. Aphids: A mobile agent-based programmable hybrid intrusion detec-tion system. In *1st International Workshop on Mobility Aware Tech-nologies and Applications (MATA 2004)*, 2004.
 - [17] R. Deraison. The nessus attack scripting language reference guide, 2000.
 - [18] Jose Duarte de Queiroz, Luiz Fernando Rust da Costa Carmo, and Luci Pirmez. Micael: An autonomous mobile agent system to protect new generation networked applications. In *2nd Annual Workshop on Recent Advances in Intrusion Detection*, 1999.
 - [19] S.T. Eckmann, G. Vigna, and R.A. Kemmerer. STATL: An Attack Language for State-based Intrusion Detection. *Journal of Computer Security*, 10(1/2):71–104, 2002.

-
- [20] Robert F. Erbacher and Deborah Frincke. Visual behavior-characterization for intrusion and misuse detection. In *Proceedings of SPIE Conference on Visual Data Exploration and Analysis VIII*, pages 210–218, 1 2001.
 - [21] Noria Foukia, David Billard, and Juergen Harms. Computer system immunity using mobile agents. In *HP Openview University Association 8th Annual Workshop*, 2001.
 - [22] Noria Foukia, Salima Hassas, Serge Fenet, and Paul Albuquerque. Combining immune system and social insect metaphors: A paradigm for intrusion detection and response system. In *Proceedings of the 5th International Workshop for Mobile Agents for Telecommunication Applications*, 2003.
 - [23] A. Fuggetta, G.P. Picco, and G. Vigna. Understanding Code Mobility. *IEEE Transactions on Software Engineering*, 24(5):342–361, May 1998.
 - [24] Guy Helmer, Johnny S. K. Wong, Vasant Honavar, and Les Miller. Intelligent agents for intrusion detection. In *Proceedings of the IEEE Information Technology Conference*, pages 121–124, 1998.
 - [25] Wayne Jansen and Ton Karigiannis. Mobile agent security, 1999.
 - [26] Samuel T. King and Peter M. Chen. Backtracking intrusions. *ACM Trans. Comput. Syst.*, 23(1):51–76, 2005.
 - [27] Christopher Kruegel and Thomas Toth. Sparta – a mobile agent based intrusion detection system. In *Proceedings of the IFIP Conference on Network Security (I-NetSec)*, 2001.
 - [28] Christopher Kruegel and Thomas Toth. Flexible, mobile agent based intrusion detection for dynamic networks. In *Proceedings of the European Wireless*, 2 2002.
 - [29] Chunsheng Li, Qingfeng Song, and Chengqi Zhang. Ma-ids architecture for distributed intrusion detection using mobile agents. In *Proceedings of the 2nd International Conference on Information Technology for Application (ICITA 2004)*, 2004.
 - [30] Richard Lippmann, David Fried, Isaac Graf, Joshua Haines, Kristopher Kendall, David McClung, Dan Weber, Seth Webster, Dan Wyschogrod, Robert Cunningham, and Marc Zissman. Evaluating intrusion detection systems: The 1998 DARPA off-line intrusion detection evaluation.

-
- In *Proceedings of the DARPA Information Survivability Conference and Exposition*, Los Alamitos, CA, 2000. IEEE Computer Society Press.
- [31] Zhengding Lu, Chunlin Li, and Layuan Li. Coordinating mobile agents by the xml-based tuple space. *J. Comput. Sci. Technol.*, 17(6):888–900, 2002.
 - [32] Roy A. Maxion and Kymie M. C. Tan. Benchmarking anomaly-based detection systems. In *Proceedings of the 2000 International Conference on Dependable Systems and Networks (formerly FTCS-30 and DCCA-8)*, pages 623–630. IEEE Computer Society, 2000.
 - [33] John McHugh. Testing intrusion detection systems: a critique of the 1998 and 1999 darpa intrusion detection system evaluations as performed by lincoln laboratory. *ACM Trans. Inf. Syst. Secur.*, 3(4):262–294, 2000.
 - [34] Peter Mell and Mark McLarnon. Mobile agent attack resistant distributed hierarchical intrusion detection systems. In *Second International Workshop on Recent Advances in Intrusion Detection*, 9 1999.
 - [35] Jelena Mirkovic and Peter Reiher. A taxonomy of ddos attack and ddos defense mechanisms. *SIGCOMM Comput. Commun. Rev.*, 34(2):39–53, 2004.
 - [36] Robert J. Moorhead, Markus Gross, and Kenneth I. Joy, editors. *Case Study: Interactive Visualization for Internet Security*. IEEE Visualization 2002 Conference, IEEE Computer Science Press, 2002.
 - [37] Vernon Paxson. Bro: A system for detecting network intruders in real-time. *Computer Networks*, 31:2435–2463, 12 1999.
 - [38] Geetha Ramachandran and Delbert Hart. A p2p intrusion detection system based on mobile agents. In *ACM-SE 42: Proceedings of the 42nd annual Southeast regional conference*, pages 185–190, New York, NY, USA, 2004. ACM Press.
 - [39] M Roesch. Snort – lightweight intrusion detection system for networks. In *Proceedings of USENIX LISA '99*, 11 1999.
 - [40] Alexandr Seleznyov and Seppo Puuronen. Anomaly intrusion detection systems: Handling temporal relations between events. In *Recent Advances in Intrusion Detection*, 1999.

-
- [41] Kapil Singh and Son T. Vuong. Blaze: A mobile agent paradigm for voip intrusion detection systems. In *ICETE (2)*, pages 238–245, 2004.
 - [42] Robin Sommer and Vern Paxson. Enhancing byte-level network intrusion detection signatures with context. In *Proceedings of the 10th ACM conference on Computer and communication security*, pages 262–271. ACM Press, 2003.
 - [43] S. Stolfo, A. Prodromidis, S. Tselepis, W. Lee, D. Fan, and P.Chan. Jam: Java agents for metalearning over distributed databases. In *AAAI97 Workshop on AI Methods on Fraud and Risk Management*, 1997.
 - [44] Thomas Toth and Christopher Kruegel. Evaluating the impact of automated intrusion response mechanisms. In *Proceedings of the 18th Annual Computer Security Applications Conference (ACSAC)*, IEEE Computer Society Press, 11 2002.
 - [45] F. Valeur, G. Vigna, C.Kruegel, and R. Kemmerer. A Comprehensive Approach to Intrusion Detection Alert Correlation. *IEEE Transactions on Dependable and Secure Computing*, 1(3):146–169, July-September 2004.
 - [46] G. Vigna. Mobile Agents: Ten Reasons For Failure. In *Proceedings of the IEEE International Conference on Mobile Data Management (MDM '04)*, pages 298–299, Berkeley, CA, January 2004. Position Paper.
 - [47] G. Vigna, B. Cassell, and D. Fayram. An Intrusion Detection System for Aglets. In N. Suri, editor, *Proceedings of the 6th International Conference on Mobile Agents (MA '02)*, volume 2535 of *LNCS*, pages 64–77, Barcelona, Spain, October 2002. Springer-Verlag.
 - [48] G. Vigna, S.T. Eckmann, and R.A. Kemmerer. Attack Languages. In *Proceedings of the IEEE Information Survivability Workshop*, Boston, MA, October 2000.
 - [49] G. Vigna and R.A. Kemmerer. NetSTAT: A Network-based Intrusion Detection Approach. In *Proceedings of the 14th Annual Computer Security Applications Conference (ACSAC '98)*, pages 25–34, Scottsdale, AZ, December 1998. IEEE Press. Best Paper Award.

-
- [50] Ke Wang and Salvatore J. Stolfo. Anomalous payload-based network intrusion detection. In *Recent Advance in Intrusion Detection (RAID)*, 2004.
 - [51] Jim Yuill, Shyhtsun Felix Wu, Fengmin Gong, and Ming-Yuh Huang. Intrusion detection for an on-going attack. In *Recent Advances in Intrusion Detection*, 1999.

Appendix A

APHIDS++ API

In this section we describe only the APHIDS++ API (Application Programming Interface) that has been developed or removed from the base APHIDS system [15]. As mentioned earlier, minimal changes were made to the API in introducing the Policy Agent.

A.1 CorrelationObject API

The following are a list of modifications made to the Correlation Object class:

```
public boolean policyEngineOn();
```

This method has to be defined by a user in APHIDS++. This method returns a boolean value that defines whether the Policy Agent will be used by this correlation routine.

```
public void doCorrelation(Map initializedData);
```

This method previously did not take an argument. This method has been now redesigned to take an argument which provides initialized data from within the `initializeAgents()` method.

```
public Map initializeAgents();
```

This method allows a user to put in any temporary data during correlation routine initialization in case the user has not done it through the APHIDS++ ADCS.

```
public AgentInfo deployTrigger(CorrelationAgent ca);
```

This method has been removed from the APHIDS++ source as its work is now done using the XML ADCS. This method was used to deploy and

initialize the Trigger Agent.

A.2 Policy Agent API

The Policy Agent is a new component that has been introduced in the current system of APHIDS++. The following are the only two methods that the user has to define.

```
public Map returnSetOfRules(String Key);
```

This method allows the Trigger Agent and Correlation Agent to retrieve a set of rules stored in the Policy Agent.

```
public Map UpdateMachineLearningRules(Map arg);
```

This method allows a user to update the rules available in the Policy Agent. This method is invoked by the Correlation Agent using the `doCorrelation(Map)` method.

Appendix B

Call Hijack Simulation

This appendix provides a visual representation of the agent interaction during the detection of a VoIP call Hijack scenario.

Figure B.1: A user has to first enter the web address where the class files can be found

Figure B.2: A user can enter the `<list>` command to see if the XML file was read correctly

Figure B.3: A user has to enter the enable `<correlationRoutine>` command to start the correlation task

Figure B.4: This shows that correlationObjects abstract class `policyEngineOn()` has been executed. This is an important class since if the return value is true, then the Policy Agent is enabled, else it is not

Figure B.5: The user now has to enter the location where the Policy Agent will reside

Figure B.6: The user now has to enter the location of the Trigger Agent which sits in the machine where MG (media gateway) and MG.log exists

Figure B.7: Enter the location of where the MG.log file exists

Figure B.8: The user now has to enter the location of the Task Agent which will move to the machine where MGC (media gateway controller) and MGC.log exists

Figure B.9: Enter the location where the MGC.log file exists

Figure B.10: A request is made to the user to allow B.9 and B.10 to take place before he/she can click on OK. This is because step B.8 happens in parallel to B.10. B8 and B9 have to execute first for the system to function properly

Figure B.11: The user has to enter the simulated orphaned packet information. For the system to work properly, the SrcIP has to exist in the allowed list of the Policy Agent. The SrcIP cannot exist in MGC.log (which keeps track of active connections), since that would mean that it is not an orphaned packet. The destinationIP has to exist in MGC.log with the REINVITE message flag

Figure B.12: The Trigger Agent is contacting the Policy Agent to check if the sourceIP of the orphaned packet is in the allowed list of IPs that can use this MG

Figure B.13: A message is displayed to the user mentioning that this is a valid source IP for an orphaned packet. The trigger agent is now going to conduct a correlation operation to find the user who sent the REINVITE message

Figure B.14: The correlation Agent will make a call to the MachineLearning component of the Policy Agent. We do this step just to check if we can contact the MachineLearning component

Figure B.15: This message reconfirms that the correlation Agent successfully contacted policy Agents Machine Learning component

Figure B.16: At this point the correlation Routine has already contacted the Task Agent to move to the MGC to get relevant data from MGC.log regarding the intrusion. Once it gets the data from the Task agent, it contacts the policy Agent to return the profile of the intruder. The correlation routine then forwards the profile to the action agent to display the information on the console

Figure B.17: The final message shows the results of the correlation process in the UI